Pseudo Random Generators

Applied Cryptography
Andrei Bulatov

One-Time Pad

- The one-time pad is the following cryptosystem (K, E, D):
  - $k \leftarrow K$ uniformly at random from $\{0,1\}^m$
  - the set of possible plaintexts is $\{0,1\}^m$
  - $E: \{0,1\}^m \times \{0,1\}^m \rightarrow \{0,1\}^m$

    $P = P^1 \dots P^m$, $k = k^1 \dots k^m$

    $C = C^1 \dots C^m$, where $C^i = P^i \oplus k^i (mod\ 2)$
  - $D: \{0,1\}^m \times \{0,1\}^m \rightarrow \{0,1\}^m$

    $P^i = C^i \oplus k^i (mod\ 2)$

Pseudorandom Generators

- The OTP is impractical because it requires a long random key. To make practical, replace it with a “pseudo-random” key
- A pseudorandom generator is a function $g: \{0,1\}^{\ell} \to \{0,1\}^{n}$
- The idea is it is given a “seed” that it extends to a long sequence of bits that sort of looks like random
- Good PRG:
  - $n \gg \ell$. The number $n - \ell$ is called the stretch of the generator
  - g is deterministic and can be efficiently computed
  - g is pseudorandom

Indistinguishability

- Let W_1, W_2 be two distributions on $\{0,1\}^n$.
- Distributions W_1, W_2 are said to be computationally indistinguishable if for any efficient statistical test Eve

$$\left|\Pr_{X \leftarrow W_1}[Eve(X) = 1] - \Pr_{X \leftarrow W_2}[Eve(X) = 1]\right| < \varepsilon$$

where $\Pr_{X \leftarrow W_i}$ means that X is sampled from W_i, and ε is negligible.

- ε is often called the advantage of the test

Pseudorandomness

g: # outputs $= 2^{\ell} \Rightarrow |U_n| >> |U_\ell|$
since g deterministic

random

- Ideally, we would like the output of a PRG to be $\underline{U_n}$, which is impossible.
- Instead, we try to make a PRG g such that $g(U_\ell)$ looks like U_n
- More precisely, $g: \{0,1\}^\ell \to \{0,1\}^n$ is pseudorandom if $g(U_\ell)$ and U_n are computationally indistinguishable
- As before we need to be conscious that what important in this definition is how all the parameters (efficiency, advantage) change as ℓ and n grow.

公理

- **PRG Axiom**: A good PRG exists

We don't know if a PRG exists

Stream Ciphers 流密码

- Let g be a pseudorandom generator producing, given a seed of length ℓ, bit strings of length n

- Let (K,E,D) be a SES defined as follows:

  K – draws keys uniformly at random from U_ℓ (i.e. a ℓ-bit string)

  E – to encrypt a plaintext P of length n it applies g to the key k and computes

$$C_i = \big(g(k)\big)_i \oplus P_i$$

  D – same as E

not used in practice

Candidate PRGs: Blum – Blum - Shub BBS

- This is a PRG that given an input of length 2ℓ produces a string of bits of length n, where n is as big as we want

- Input: an ℓ-bit integer N and integer X, $1 \le X \le N$

```
num_outputted = 0;
while num_outputted < n:
        X := X*X mod N
        num_outputted := num_outputted + 1
        output (least-significant-bit(X) )
endwhile
```

Blum – Blum – Shub is Good

- **Theorem.**

  The BBS PRG is pseudorandom if the assumption below is true

- **Assumption.**

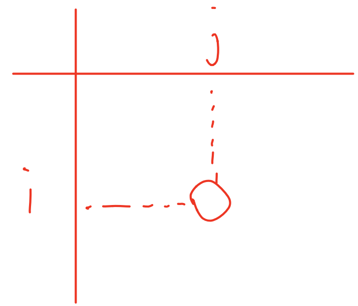
  There are superpolynomial T, ε such that for any randomised algorithm Alg with time complexity less than T the following holds

  $\Pr[\text{Alg finds factorization of a random } n\text{-bit integer}] < \varepsilon(n)$

use " int factorization is hard "

Candidate PRGs: RC4

- RC4 stands for Ron's Cipher no. 4
- Widely used: SSL (and then TLS), SSH, WEP, WPA (IEEE 802.11), BitTorrent protocol encryption, Microsoft Point-to-Point Encryption,
- A byte is a number from {1,…,256}
- Input: a permutation S: {1,…,256} → {1,…,256}



i := 0 j := 0

while num_outputted < n :

i := (i + 1) mod 256 j := (j + S[i]) mod 256

swap(S[i],S[j]) 每次都稍微改下 permutation，防重复，更随机

output (S[(S[i] + S[j]) mod 256])

endwhile

Candidate PRGs: RC4 (cntd)

8: 1 byte = 8 bits

256: 256个不同的 bytes

8×256
$\parallel$

- RC4 given an input of length 2048 produces an output of length n, which is as big as we want
- If 2048 is too much, there is another algorithm – KSE, the Key Scheduling Algorithm – that uses an input of length $40 \leq \ell \leq 128$ to generate S

  key length

- Input: a key k of length ℓ, $40 \leq \ell \leq 128$

```
for i from 0 to 255     S[i] := i     endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + k[i mod n]) mod 256
    swap(S[i],S[j])
endfor
```

Better Pseudorandom Generators

- RC4 has a number of known weaknesses (will be discussed later)
- Not recommended for use in new applications
- Better PRGs include Yarrow, Fortuna, CryptGenRandom, ChaCha20 and others

PRG implement diff in HW & SW

JAVA Pseudorandom Generator

- JAVA uses a linear congruential pseudorandom generator

```
synchronized protected int next(int bits) {
    seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
    return (int)(seed >>> (48 - bits));
}
```

Java PRG is statistical, but not pseudorandom

- It is not good for Crypto!

triple num
pair num

- Two kinds of PRG

eg. a more uniform distribution on 1 num

  – Statistical – used for computational purpose, a good statistic property
  – Cryptographic – RC4, BBS

Unpredictability

- Function $g: \{0,1\}^\ell \rightarrow \{0,1\}^n$ is predictable if there is an efficient algorithm Alg and $i < n$ that given the first i bits of $g(k)$ predicts $i + 1$st bit with high probability
- More precisely,

$$\Pr\left[\text{Alg}\left(g(k)_{1,\ldots,i}\right) = g(k)_{i+1}\right] > \frac{1}{2} + \varepsilon$$

for some non-negligible ε

- Function is unpredictable if it is not predictable. That is, for any efficient algorithm and any i ….
- Example: Linear congruential generator $x_{i+1} \equiv a \cdot x_i + b$

↳ predictable

Unpredictability II

- Function $g: \{0,1\}^\ell \to \{0,1\}^n$ is unpredictable if and only if it is pseudorandom
- Indeed, if g is predictable by an algorithm Alg, we can distinguish $g(k)$ from U_n by running Alg on the first i bits and try to predict bit $i+1$. If successful, we conclude we are dealing with $g(k)$, otherwise we conclude it is U_n. As g is predictable, we have non-negligible advantage
- The other way is a bit trickier